

Applications of Genetic Algorithms and Quantum
Genetic Algorithms on theoretical fully-autonomous
road systems

University of Birmingham



Sam Barrett
sjb786@student.bham.ac.uk

November 24, 2020

Contents

1	Introduction	4
2	Background	5
2.1	Genetic Algorithms	5
2.1.1	History	5
2.1.2	Definition	5
2.1.3	Selection	6
2.1.4	Variation	8
2.1.5	Evaluation	9
2.1.6	Reproduction	9
2.1.7	Bézier Curves	9
2.2	Fully Autonomous Road Networks	10
2.3	Quantum Genetic Algorithms	10
2.4	Alternative Technologies	10
3	Literature Review	11
3.1	Classical GAs	11
3.2	Fully Autonomous Road Networks	11
3.3	Quantum GAs	11
3.3.1	Quantum Computing	11
4	Classical Approach	12
4.1	Approach	12
4.2	Implementation	12
4.2.1	Language Choice	12
4.3	Results	13
5	Quantum Approach	14
5.1	Approach	14
5.2	Implementation	14

5.3	Results	14
6	Evaluation	15
7	Conclusion	16

Abstract

Chapter 1

Introduction

Chapter 2

Background

2.1 Genetic Algorithms

2.1.1 History

Genetic algorithms are optimisation techniques that employ the same rationale as classical Evolution seen in nature.

Genetic Algorithms can trace their origins back to the late 1960s when they were first proposed by John Holland, though he then referred to them as *Genetic Plans*¹. Holland went on to write the first book on the subject titled *Adaptation in Natural and Artificial Systems*[1] in 1975. The field did not find much reception with Holland stating in the preface to the 1992 rerun:

“When this book was originally published I was very optimistic, envisioning extensive reviews and a kind of ‘best seller’ in the realm of monographs. Alas! That did not happen.”

However, in the early nineties, Genetic algorithms surged in popularity along with the area of Artificially Intelligent planning as a whole leading to Holland republishing his book and solidifying his position as the field’s founder.

2.1.2 Definition

In a general sense, optimisation techniques work to find the set of parameters \mathcal{P} that minimise an objective function \mathcal{F} . Genetic algorithms approach this

¹This distinction was made to emphasise the “centrality of computation in defining and implementing the plans”[1]

by representing these sets as individuals in a population, P . Over the course of multiple generations, the best solutions are determined and promoted until termination criteria are met or the maximum number of generations is reached.

As our candidates are essentially a collection of parameters to the function we are trying to optimise, we can extend our metaphor further by mapping each element of an individual to a *gene* in an individual's genome.

The representation we use in a GA is problem specific. Often we have to provide functions to facilitate the mapping between the problem specific set of possible solutions and the encoded genotype space in which we optimise. The most basic representation being a string of binary numbers.

Genetic algorithms are both *probabilistically optimal* and *probabilistically complete*[2] meaning that: given infinite time, not only will the algorithm find a solution, (if one exists), it will find **the** optimal solution from the set of all possible solutions, \mathcal{P}^* .

Algorithm 1: Modern Generic Genetic Algorithm

Result: Best Solution, p_{best}
Generate initial population, P_0 of size n ;
Evaluate fitness of each individual in P_0 , $\{F(p_{0,1}, \dots, p_{0,n})\}$;
while *termination criteria are not met* **do**
 Selection: Select individuals from P_t based on their fitness;
 Variation: Apply variation operators to parents from P_t to produce offspring;
 Evaluation: Evaluate the fitness of the newly bred individuals;
 Reproduction: Generate a new population P_{t+1} using individuals from P_t as well as the newly bred candidates.;
 $t++$
end
return p_{best}

As you can see from Figure 2.1 and Algorithm 1 the overall shape of GAs has not changed substantially over the course of the past 50 years. Being comprised of a series of operations that a starting population is piped through until criteria are met.

2.1.3 Selection

The selection procedure is the process by which the next generation of individuals is created from the current population. Individuals are selected relative to their fitness as determined by the Objective (fitness) function \mathcal{F} .

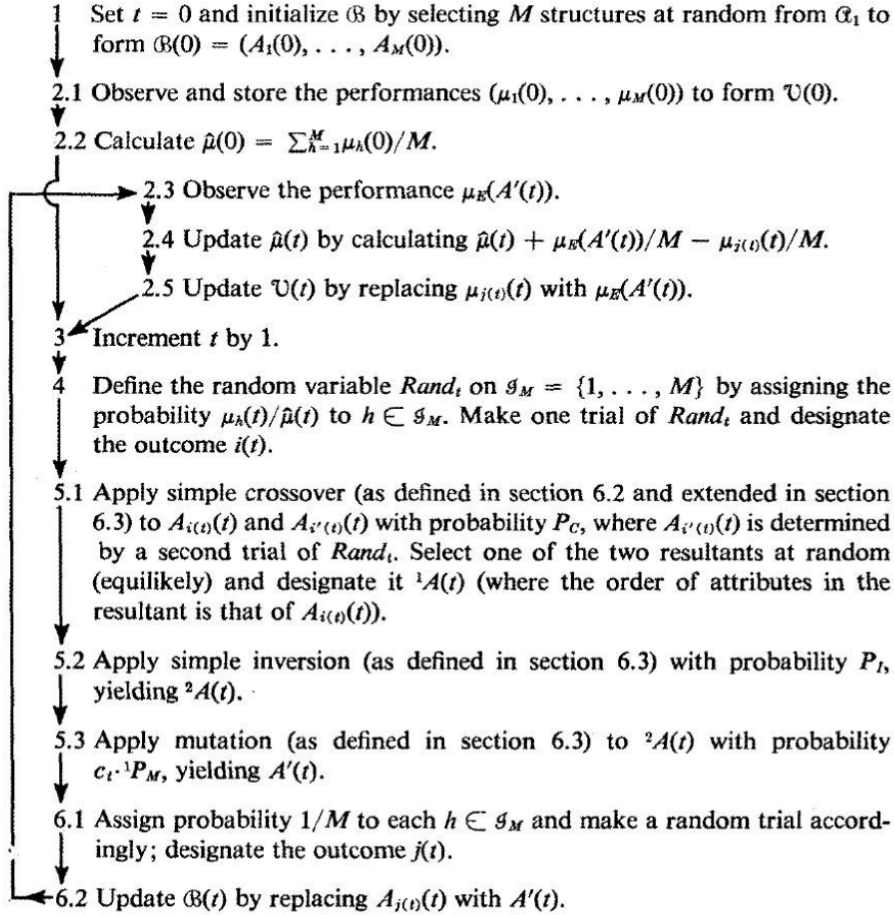


Figure 2.1: GA algorithm outlined in Holland's Original Book[1]

Some methods select only the best solutions by fitness. Others employ a more stochastic approach such as the Monte Carlo method[3] to increase diversity and reduce complexity.

2.1.4 Variation

Variation in a GA is the process of altering the genome individuals to further explore the search space via stochastic local search. We perform this using two distinct sub-operations: Mutation and Crossover.

Mutation

In mutation we alter each gene with a set probability p_m known as the *mutation rate*. A standard value for a mutation rate is $\frac{1}{L}$ but it can fall anywhere in the range $p_m \in [\frac{1}{L}, \frac{1}{2}]$ where L is the length of the genome. In binary coded GAs we alter a given gene by *flipping* it's value. In real-coded GAs mutation operators include:

- Uniform Mutation
- Non-Uniform Mutation
- Gaussian Mutation

A low value for p_m a new individual which can be shown to be *close* to it's parents in the search space relative to their *Hamming distance*² if using a Binary coded GA. In real-coded GAs they can be shown to be close by the Euclidean distance between them.

Crossover

Crossover is a binary operation, taking two randomly selected *parents* from the population P_t with a probability $p_c \in [0, 1]$

There are two major forms of crossover for binary coded GAs: $n > 1$ point crossover and Uniform crossover.

For real-coded GAs you instead have a selection of Crossover operators including:

- Flat crossover
- Simple crossover

²Hamming Distance: The metric for comparing two binary data strings. The Hamming distance between two strings is the number of bit position in which they differ.

- Whole arithmetic crossover
- Local arithmetic crossover
- Single arithmetic crossover
- BLX- α crossover

2.1.5 Evaluation

After developing potential new individuals, the fitness of these new individuals is calculated using the objective function, \mathcal{F}

2.1.6 Reproduction

Here the next generation of individuals is constructed. There are multiple potential methods employed here with the simplest being to just replace the least fit individuals with additional copies of the fitter individuals, be they pre-existing or newly generated.

In this stage various heuristics or alternative strategies can be implemented to speed up or slow down the convergence rate. Whilst their fitness may be low, having a diverse population allows for more of the search space to be explored. If the algorithm converges too quickly it may get *stuck* in local minima.

2.1.7 Bézier Curves

In my implementation, I utilise Bézier curves to encode complex route arcs between a series of points. Here I will briefly outline their construction and mathematical basis.

Bézier curves were developed by and named after French auto-body designer Pierre Bézier and are commonly found in computer graphics today. They are parametric curves made up of a series of *control points* which contort the shape of a line to produce a curve of nearly any shape.

A Bézier curve is said to have a degree of $n - 1$ where n is the number of control points including the start and end point of the curve.

Formal Definition

Bézier curves can be simply explicitly defined for degrees of 1 through 4, however, the general case of n points is more useful to us. In their general

form they can be defined recursively or explicitly. I implement the recursive version as it is much more readable for a programming language that supports recursion.

They are defined recursively as follows:

$$\mathbf{B}_{P_0}(t) = P_0 \quad (2.1)$$

$$\mathbf{B}_{P_0, P_1, \dots, P_n}(t) = (1 - t)\mathbf{B}_{P_0, P_1, \dots, P_{n-1}}(t) + t\mathbf{B}_{P_1, P_2, \dots, P_n}(t) \quad (2.2)$$

Where the parameter t is in the range of $[0, 1]$. Therefore, the smoothness of the curve is determined by the granularity of the parameter when unpacking the function into a curve.

2.2 Fully Autonomous Road Networks

Fully autonomous road networks are by no means a novel concept. They have appeared in fiction since the mid 20th century and have been a real possibility for the past decade. In a fully autonomous road networks (*FARN*)s, all vehicles are self-operating with their routing either being determined on a vehicle-by-vehicle *selfish* basis or by a larger system managing routes for all nearby agents.

In the latter system protocols that promote the net increase in efficiency can be implemented. However, this sort of system will require a form of government mandate as a universal routing protocol would need to be established; this is no small undertaking.

2.3 Quantum Genetic Algorithms

2.4 Alternative Technologies

Chapter 3

Literature Review

3.1 Classical GAs

3.2 Fully Autonomous Road Networks

FARNs have the possibility of improving travel in a number of different areas.

They have the potential of greatly reducing travel times by efficiently routing all vehicles with the goal of reducing the net travel time. Such a system would also be able to respond much faster than human drivers, allowing for large increases in permissible vehicle velocities thus, further increasing efficiency.

By extension of their higher efficiency, *FARNs* will also have a marked effect on vehicular energy consumption. The potential effects are summarised nicely by Vahidi and Sciarretta[4] where they show the use of vehicle automation could cause anywhere from a halving of "energy use and greenhouse gas emission" in an "optimistic scenario" to doubling them depending on the effects that present themselves. The increase in highway speed whilst increasing travel efficiency is predicted by Brown et al.[5] and Wadud et al.[6] as increasing energy use by anywhere from 5% to 30%

3.3 Quantum GAs

3.3.1 Quantum Computing

Chapter 4

Classical Approach

4.1 Approach

The general shape of a GA as seen in Algorithm 1 is the same for almost all problems.

As such, I first had to implement a method to generate an initial population. In order to do this, I first had to define my programmatic representations of the Phenotypes and genotypes of my individuals. I decided to base my genotype structure on the one described by Kala[2]

4.2 Implementation

My for my initial implementation, I focused on simply getting a working system. I therefore, started by implementing the simplest form of each genetic operator. Namely, roulette wheel (Monte Carlo[3]) selection, simple crossover and uniform mutation.

4.2.1 Language Choice

I have chosen to implement my classical approach using the Julia language project[7].

Julia is a relatively new language first developed in 2012 by Jeff Bezanson, Stefan Karpinski and Viral Shah. It is a multi-paradigm language allowing for functional, object oriented and meta programming approaches to problems. I will mainly be using it for it's functional and OO capabilities.

Julia operates using multiple dispatch similar to languages such as Haskell. It interoperates with C and Fortran codebases without the need of middle-

man bloat. This fact allows it to utilise the extensive high performance C libraries for floating point operations. Julia is eagerly evaluated, uses a Just in time compiler and has a garbage collector.

Julia features a syntax similar to both Python and Matlab with performance on par with C. As I am already very familiar with python and have studied functional programming in a number of modules; I found this language very quick and intuitive to learn and the resulting code to be clean, idiomatic and fast.

A real-world implementation of a system based on my research would undoubtedly be required to run on small, relatively low performance, embedded systems and as such Julia may not be appropriate here. A language such as C or Rust may be used instead.

4.3 Results

Chapter 5

Quantum Approach

5.1 Approach

5.2 Implementation

5.3 Results

Chapter 6

Evaluation

Chapter 7

Conclusion

Bibliography

- [1] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press, Cambridge, UNITED STATES, 1992.
- [2] Rahul Kala. *On-Road Intelligent Vehicles: Motion Planning for Intelligent Transportation Systems / Rahul Kala*. Butterworth-Heinemann is an imprint of Elsevier, Kidlington, Oxford, UK, 2016.
- [3] N Metropolis. The beginning. *Los Alamos Science*, 15:125–130, 1987.
- [4] Ardalan Vahidi and Antonio Sciarretta. Energy saving potentials of connected and automated vehicles. *Transportation Research Part C: Emerging Technologies*, 95:822–843, October 2018.
- [5] Austin Brown, Jeffrey Gonder, and Brittany Repac. An Analysis of Possible Energy Impacts of Automated Vehicles. In Gereon Meyer and Sven Beiker, editors, *Road Vehicle Automation*, Lecture Notes in Mobility, pages 137–153. Springer International Publishing, Cham, 2014.
- [6] Zia Wadud, Don MacKenzie, and Paul Leiby. Help or hindrance? The travel, energy and carbon impacts of highly automated vehicles. *Transportation Research Part A: Policy and Practice*, 86:1–18, April 2016.
- [7] The Julia Programming Language. <https://julialang.org/>.